Title:	Autoload: a pipeline for expanding the holdings of an Institutional Repository enabled by ResourceSync
Author(s):	Powell, James Estes Jr. Klein, Martin Van De Sompel, Herbert
Intended for:	Code4Lib Journal
Issued:	2017-04-21

Autoload: a pipeline for expanding the holdings of an Institutional Repository enabled by ResourceSync

By James Powell (0000-0002-3517-7485), Martin Klein(0000-0003-0130-2097), and Herbert Van de Sompel(0000-0002-0715-6126)

Abstract

Providing local access to locally produced content is a primary goal of the Institutional Repository (IR). Guidelines, requirements, and workflows are among the ways in which institutions attempt to ensure this content is deposited and preserved, but some content is always missed. At Los Alamos National Laboratory, the library implemented a service called LANL Research Online (LARO), to provide public access to a collection of publicly shareable LANL researcher publications authored between 2006 and 2016. LARO exposed the fact that we have full text for only about 10% of eligible publications for this time period, despite a review and release requirement that ought to have resulted in a much higher deposition rate. This discovery motivated a new effort to discover and add more full text content to LARO. Autoload attempts to locate and harvest items that were not deposited locally, but for which archivable copies exist. Here we describe the Autoload pipeline prototype and how it aggregates and utilizes Web services including Crossref, SHERPA/RoMEO, and oaDOI as it attempts to retrieve archivable copies of resources. Autoload employs a bootstrapping mechanism based on the ResourceSync standard, a NISO standard for resource replication and synchronization. We implemented support for ResourceSync atop the LARO Solr index, which exposes metadata contained in the local IR. This allowed us to utilize ResourceSync without modifying our IR. We close with a brief discussion of other uses we envision for our ResourceSync-Solr implementation, and describe how a new effort called Signposting can replace cumbersome screen scraping with a robust autodiscovery path to content which leverages Web protocols.

Introduction

The Institutional Repository (IR) is an ecosystem of open source software components that collectively support archiving, preservation, discovery, and dissemination of content. An IR aims to house the institutional output of an organization. But they often grapple with routing content into the archive in a timely, consistent, and efficient manner. This paper describes a work in progress effort to automatically discover and expand the holdings of an institutional repository. This pilot project uses ResourceSync [1] to initialize a discovery and harvesting pipeline that programmatically evaluates discovered content for relevance, associates bibliographic and publisher licensing metadata with it, and aggregates the results in a graph store.

The Research Library at Los Alamos National Laboratory uses Solr [2] as the foundation for Web based applications for end users. Metadata is retrieved from the aDORe repository [3], indexed, and then made available through Java Web applications. One of these Web applications is called LANL Research Online (LARO). It provides search and faceted browse access to LANL authored publications that have been approved for general release. LANL has long standing requirements for review and approval prior to publication or presentation of researcher content. There is now a Web workflow that makes this requirement much easier to adhere to and this will eventually improve coverage. But to date there's been no effort to retroactively populate the repository with older published content.

We discovered the extent of the coverage problem as we were laying the groundwork to publicize LARO content to search engines. We were planning to use ResourceSync to provide an iterable list of LARO publications and their metadata. ResourceSync is a NISO standard designed to facilitate content replication and synchronization. One use case for ResourceSync is to make a collection of Web resources accessible and available for discovery. LARO contains metadata for about 25,415 publications and we thought a reasonable percentage of these metadata records would reference local copies of the full text item that they described. Our plan was to use ResourceSync in conjunction with Solr to expose this collection so that it could be harvested and indexed.

When we discovered that there were only 1023 full text publications in LARO, we decided that there was not yet enough content to justify marketing it widely. Our new goal was to increase the full text in LARO and here again we thought that ResourceSync might be an excellent tool for bootstrapping this effort. To explain why, we will briefly review the features and capabilities of this standard.

ResourceSync

ResourceSync is a NISO standard [4] approved in 2014, and updated in 2017, as a solution to the problem of synchronizing Web resources between sites. It is specifically concerned with exposing the locations of the Web resources and their descriptions. To accomplish this, ResourceSync defines a framework of synchronization capabilities. A content provider is referred to as the source, while a site that intends to replicate the source's content is called a destination. ResourceSync defines modular capabilities which can be implemented by the source to enable a destination to retrieve and verify the integrity of the source's content as it changes over time. Capabilities supported by the ResourceSync framework include “pull” and “push” baseline synchronization and incremental synchronization, bulk synchronization, and audit/verification, for metadata and content. ResourceSync specifies an XML vocabulary for describing ResourceSync framework capabilities implemented by the source, and for describing the resources and metadata available for replication. This adds a few element and attribute extensions to the Sitemap protocol [5]. Sitemap is an XML specification for describing and itemizing the various pieces of a Web site, for discovery, navigation, and indexing. ResourceSync builds on sitemap [6] with elements that make it possible for a destination to make a copy of a site's content, find the associated metadata for that content, find information about changes that have occurred at a source over time, and data that can be used to verify the integrity of the destination's copy of a source.

ResourceSync is modular [7], which means that a source may be selective about which parts of the standard it implements. A simple approach is to enable destinations to stay in sync via pull requests. The source publishes a baseline list of resources (a resource list) as well as lists of updates as the site changes over time (change lists). But a source may elect to use other parts of the standard. For example, if the source collection changes frequently, then it could provide push change notifications to decrease synchronization latency. Or if the source wants to support bulk replication for a large amount of content, then it could publish resource dumps and change dumps, which are collections of content and their metadata bundled together and compressed for easier transfer to a destination. Any type of resource is supported by the framework as long as it can be accessed via a URI. A capability list describes which parts of the ResourceSync framework a source implements.

In our case, LARO functions as a source, and the Autoload pipeline is a destination. LARO supplies a resource list for baseline synchronization, and change lists which contain new entries when new items are indexed in LARO. So the Autoload process starts by retrieving a resource list or change list.

ResourceSync and Solr

Many IRs, including Fedora/Hydra and Duraspace/DSpace use Solr. Solr is an open source application that provides indexing and searching functionality which utilizes the Apache Lucene search library. Lucene is a mature platform for textual search, and Solr adds simple forms-based support for indexing textual elements of structured content, such as metadata. Solr can be used to provide a variety of field-specific, cross-element, value-bounded, and faceted queries. This functionality is available not only to end users but also to applications through a REST interface. At the Research Library, our metadata records reside in our aDORe repository. aDORE uses the MPEG-21 Digital Item Declaration Language (DIDL) as a container for metadata and digital objects. The DIDL is referenceable via a digital item identifier (DID). In this way, aDORE can accommodate any type of digital objects and any type of metadata. We store the MARC XML file inside the DIDL. For many end-user applications, we pull this MARC XML metadata from the repository and index this data with Solr.

Although LARO is technically our ResourceSync source, we actually implemented the source as an intermediary, external to LARO. It queries LARO's Solr instance to generate ResourceSync lists. We did this for practical reasons – we did not want to modify the repository codebase or any existing applications in order to support ResourceSync. LARO is a virtual collection which has no explicit identification within the repository. Metadata resides in the aDORe repository and it is extracted according to certain selection criteria, indexed with Solr and made available under the LARO banner. So effectively, the Solr index that supports LARO is the only record of membership for the LARO collection.

Solr provides a convenient interface upon which to build ResourceSync capabilities for a couple of reasons.

- First, it has two characteristics which are essential for building "pull" synchronization components – unique Web referenceable identifiers, and timestamps. Timestamps are necessary in order to provide temporal details about the assembled collection of URIs in the resource list and the change lists to destinations.

- Second, Solr provides a REST interface for resource URIs which the source intermediary can query, thus avoiding the need to build new hooks into the repository or touch any existing code associated with the repository or LARO.

```
from resync import Resource,ResourceList
rl = ResourceList()
thisMoment = str(datetime.now())
resourcelist_timestamp = thisMoment
rl.up = "http://example.com/dataset1/capabilitylist.xml"
rl.md_until = resourcelist_timestamp
  thisResource = Resource(uri=linkVal, lastmod = timestamp)
  thisResource.link_set(rel="describedby", modified = timestamp, href =
recMetadataUri)
  thisResourceRecip = Resource(uri = recMetadataUri, lastmod = timestamp)
  thisResourceRecip.link_set(rel="describes", href=linkVal, modified=timestamp)
  thisResourceRecip.link_set(rel="profile",
href="http://www.w3.org/2001/XMLSchema-instance")
  rl.add([thisResource, thisResourceRecip])
```

Listing 1: using the resync library to create resource list entries

Overview of the solrSync application

Various software libraries exist for implementing ResourceSync capabilities including the Python `resync` library [8] for building and testing various ResourceSync framework capabilities, and `rspub-core`[9] which is a comprehensive, extensible framework for adding ResourceSync capabilities to an existing Web application, repository, or database via plug-ins. There is also a library for push notifications called `resourcesync_push` [10] (based on `PubSubHubbub`). Our implementation, which we refer to as solrSync, was implemented in Python using the `resync` library, atop the Django Web framework [11]. Our goal was to implement resource lists and change lists as dynamic elements constructed from the LARO Solr index. For this test collection which contains around 25,000 items, this approach seems to work quite well. The solrSync prototype's codebase is quite modest and demonstrates that ResourceSync is easy to support with a very small amount of effort (see listing 1). No existing systems had to be modified to support ResourceSync, demonstrating that ResourceSync capabilities can be easily added to an existing IR ecosystem.

Here are the steps that are performed when a resourcelist.xml is requested from solrSync:

1) When a request is made for resourcelist.xml, the Django solrSync application resolves this via a pattern for resourcelist.xml in its `urls.py` file
2) This triggers the execution of a method which queries Solr for a set of results up to a given date
3) It parses the Solr results set and locates a resumption token which occurs as the text node of a `str` element with an attribute `name` that contains the value "`nextCursorMark`"
4) It iterates through the `doc` element blocks in the Solr results and for each, it locates the `str` element that has a name attribute with the value "`recID`" (this is specific to our instance of Solr)
5) It uses a base URI value to create a direct reference to this item's metadata in the aDORe repository
6) It constructs a `resync` Resource item based on this data
7) It examines each Solr result entry for a reference to a full text report or other full text content, and if it exists, adds this to the resource list as well (see below example). These are items that the Autoload pipeline will skip as it consumes the resource list.
8) It repeats this process from step 2 onward until it exhausts the Solr result set
9) Finally, solrSync outputs XML for the resource list and returns it to the requester

There are several important considerations for building a resource list or change list from a Solr results set. A Solr result set includes an important piece of information that becomes crucial when dealing with large numbers of items: the `numFound` attribute. The sitemap protocol limits the number of URIs it can reference to 50,000. If `numFound` exceeds this value directly, or when combined with references to both full text and metadata, then it is necessary to generate multiple resource list or change list files. These are itemized in an index referenced from the capability list, which could also be dynamically generated. If this information cannot be computed, then it is not possible to generate resource lists and change lists that conform to the Sitemap protocol.

The solrSync approach will not work if the Solr index does not include a meaningful timestamp per item indexed. This value must be searchable, and it must be returned as part of the result set. This approach will also fail if the Solr index does not contain and does not return the identifiers for indexed items. This is generally not a problem with bibliographic metadata indexes since the point of the Solr index is to facilitate content discovery. Listing 2 provides an example Solr query parameter list for solrSync:

```
q=title:* AND timestamp:[* TO 2017-02-08T21:18:33Z]
&wt=xml&indent=true&sort=recID+asc&cursorMark=*
```

Listing 2: Solr query example

This query, which is used to generate a resource list, retrieves all records that have a timestamp up to and including the value specified, returns the results as XML, sorts on the unique identifier "`recID`", and indicates that the result set should be paged using cursormark.

The timestamp parameter for a Solr query for a change list is slightly different than for a resource list. In addition to specifying a date range, it uses curly brackets to indicate that the result set should contain items that were added or changed since the value specified until the present moment, as illustrated in listing 3:

```
timestamp:{2016-08-20T16:33:33Z TO NOW}
```
Listing 3: Alternate temporal parameter for Solr query to generate a change list

solrSync uses the Python requests library to submit a query to Solr via HTTP, and then uses the Python library `xmldict` to load and parse [12] the Solr result set (listing 4). The Solr results are transformed into a ResourceSync resource list or change list. Since most of the LARO collection is represented only by metadata, the LARO resource list contains mostly metadata URIs, which point back to the aDORe repository (listing 6). When there is full text available, this information is included in the LARO Solr index. The index will contain a reference to a permalink, a shareable and bookmarkable URL that can be derefenced to access full text content on a local content server. When full text does exist for a LARO record, two ResourceSync loc elements are included for this Solr result entry. An example is provided in listing 5. The ResourceSync `rs:ln` is a child element of the loc element that contains a URI. This URI has a relationship with the resource indicated by the `rel` attribute. If the `rel` attribute value is "describes" then the rs:ln URI is for the content. If the rel attribute value is "`describedby`" then the `rs:ln` URI is a pointer to metadata about the content. solrSync generates a resource list or change list using the `resync` Python library, which builds a Python resourcelist object in memory, and then outputs it as xml. The `resync` library can be used to generate other documents for ResourceSync such as a capability list, which contains links to a source's resource lists and change lists.

```
<response>
<lst name="responseHeader">
  <int name="status">0</int>
  <int name="QTime">83</int>
  <lst name="params">
    <str name="sort">recID asc</str>
    <str name="indent">true</str>
    <str name="q">title:* AND timestamp:[* TO 2016-07-05T23:23:54Z]</str>
    <str name="wt">xml</str>
    <str name="rows">25500</str>
  </lst>
</lst>
<result name="response" numFound="25415" start="0">
...
  <doc>
    <str name="recID">info:lanl-repo/lareport/LA-UR-14-29153</str>
    <str name="dataset">RASSTI</str>
    <str name="displayName">Guzik, Joyce Ann ; Bradley, Paul Andrew ; Jackiewicz, 
Jason ; Molenda-Zakowicz, Joanna ; Uytterhoeven, Katrien ; et al. </str>
    <str name="displayTitle">The Occurrence of Non-Pulsating Stars in the gamma Dor 
and delta Sct Pulsation Instability Regions: Results from Kepler Quarter 14-17 
Data</str>
    <arr name="publication">
      <str>Astronomical Review</str>
    </arr>
    <arr name="publication_browse">
```

```
      <str>Astronomical Review</str>
    </arr>
    <arr name="publication_rbrowse">
      <str>Zhgilmlnrxzo Ivervd</str>
    </arr>
    <str name="doi">10.1080/21672857.2015.1023120</str>
    <str name="displaySource">Astronomical Review ; Vol.9, iss.1, p.41-65, Jan.
2014</str>
    <str name="openaccess">true</str>
    <str name="afv_flag">true</str>
    <str name="embargodate">2015-04-21</str>
    <arr name="url">
      <str>URL Accepted Manuscript | http://arxiv.org/abs/1403.8013</str>
      <str>Accepted Manuscript | http://permalink.lanl.gov/object/AMpdf?
what=info:lanl-repo/lareport/LA-UR-14-29153</str>
    </arr>
    <long name="_version_">1522819486080565248</long>
    <date name="timestamp">2016-01-08T17:23:52.373Z</date>
   </doc>
  </result>
</response>
```
Listing 4: Solr results excerpt

```
<url>
  <loc>http://lastage.lanl.gov:8080/adore-disseminator/service?url_ver=Z39.88-
2004&amp;rft_id=info:lanl-repo/lareport/LA-UR-14-29153&amp;svc_id=info:lanl-
repo/svc/xml.format.full</loc>
  <lastmod>2016-01-08T17:23:52.373000Z</lastmod>
  <rs:ln href=" http://permalink.lanl.gov/object/AMpdf?what=info:lanl-
repo/lareport/LA-UR-14-29153" modified="2016-01-08T17:23:52.373Z"
rel="describes" />
  <rs:ln href="http://www.w3.org/2001/XMLSchema-instance" rel="profile" />
</url>

<url>
  <loc> http://permalink.lanl.gov/object/AMpdf?what=info:lanl-repo/lareport/LA-UR-
14-29153</loc>
  <lastmod>2016-01-08T17:23:52.373000Z</lastmod>
  <rs:ln href="http://lastage.lanl.gov:8080/adore-disseminator/service?
url_ver=Z39.88-2004&amp;rft_id=info:lanl-repo/lareport/LA-UR-14-
29153&amp;svc_id=info:lanl-repo/svc/xml.format.full" modified="2016-01-
08T17:23:52.373Z" rel="describedby" />
</url>
```
Listing 5: resource list entry corresponding with the Solr result from listing 4

```
<record>
   <title>The Occurrence of Non-Pulsating Stars in the gamma Dor and delta Sct
Pulsation Instability Regions: Results from Kepler Quarter 14-17 Data</title>
   <authors>Guzik, Joyce Ann (XTD-NTA: XTD NUCLEAR THREAT ASSESSMENT)
(joy@lanl.gov)  ; Bradley, Paul Andrew (XCP-6: PLASMA THEORY AND APPLICATIONS)
(pbradley@lanl.gov)  ; Jackiewicz, Jason (New Mexico State U., Las Cruces, NM)  ;
Molenda-Zakowicz, Joanna (Uniwersytet Wroclawski, Wroclaw, Poland)  ; Uytterhoeven,
Katrien (Instituto de Astrofisica de Canarias, Tenerife, Spain)  ; Kinemuchi, Karen
(Apache Point Observatory, Sunspot, NM) </authors>
   <institutions>XTD-NTA: XTD NUCLEAR THREAT ASSESSMENT ; XCP-6: PLASMA THEORY AND
APPLICATIONS ; New Mexico State U., Las Cruces, NM ; Uniwersytet Wroclawski,
Wroclaw, Poland ; Instituto de Astrofisica de Canarias, Tenerife, Spain ; Apache
Point Observatory, Sunspot, NM</institutions>
   <source>Astronomical Review ; Vol.9, iss.1, p.41-65, Jan. 2014</source>
   <pubDate>2015-02-09</pubDate>
   <osti_subjects>Astronomy &amp; Astrophysics(79)</osti_subjects>
   <keywords>variable stars ; stellar pulsation ; Kepler spacecraft</keywords>
   <document_type>Journal Article</document_type>
   <report_number>LA-UR-14-29153</report_number>
   <pages>29 p.</pages>
   <sponsor>NASA</sponsor>
   <openaccess>true</openaccess>
   <issn>2167-2857</issn>
   <doi>10.1080/21672857.2015.1023120</doi>
   <lapr_id>LAPR-2014-020948</lapr_id>
   <accepted_manuscript>true</accepted_manuscript>
   <program_code>5J880A-RKWC; 5J880A-RMDB</program_code>
   <restrictions>This report has access restrictions. See Los Alamos Authors
database record for details.</restrictions>
   <repository_id>info:lanl-repo/lareport/LA-UR-14-29153</repository_id>
   <dataset>RASSTI</dataset>
</record>
```
Listing 6: aDORe disseminator XML metadata serialization for URI in <loc> element from listing 5

The Autoload Pipeline

The goal of the Autoload pipeline (figure 1) was to establish a self-sustaining pipeline that would monitor LARO and update it with full text content. Feeding the pipeline with ResourceSync would allow it to function as a "machine that would go of itself." The overall effort is similar in its goals to that described by Flynn et al, 2013 [13]. The authors describe a two phase approach to populating an IR – first harvesting, transforming and ingesting metadata, and second, retrieving licensing information from SHERPA/RoMEO [14] to aide staff efforts to manually populate their IR with full text content. Their goal was to make populating the IR less staff intensive. Our effort diverges markedly in many respects, and these differences stem from our effort to use the ResourceSync standard together with various Web services to develop a self-sustaining automated pipeline to populate our IR.

Functioning as a ResourceSync destination, the first thing the Autoload pipeline does is request a resource list to establish a baseline, or a change list upon subsequent executions, which will provide information about changes that occurred since the baseline was established. As noted above, these are XML documents which itemize a list of resources available via HTTP. Autoload parses this document using the Python `xmltodict` library. We are interested in the items that lack full text, so when a resource list or change list `loc` element has only a metadata link, Autoload injects this into the pipeline. When this file contains `<loc>` elements that have associated full text content (reciprocal `describes` and `describedby` relationships) , they are skipped. The pipeline formulates a request to the aDORe repository disseminator for the metadata pointed to by that URI. In the aDORe architecture, disseminators have the ability to generate different serializations of the archived metadata. Autoload requests a basic XML serialization which includes a reference to the object's DOI, if known. Listing 6 illustrates an XML metadata record from the repository. This DOI (contained within the `<doi>` element) is used in subsequent steps in the pipeline. Only items that have associated DOIs, and which currently lack full text, are processed by the Autoload pipeline.

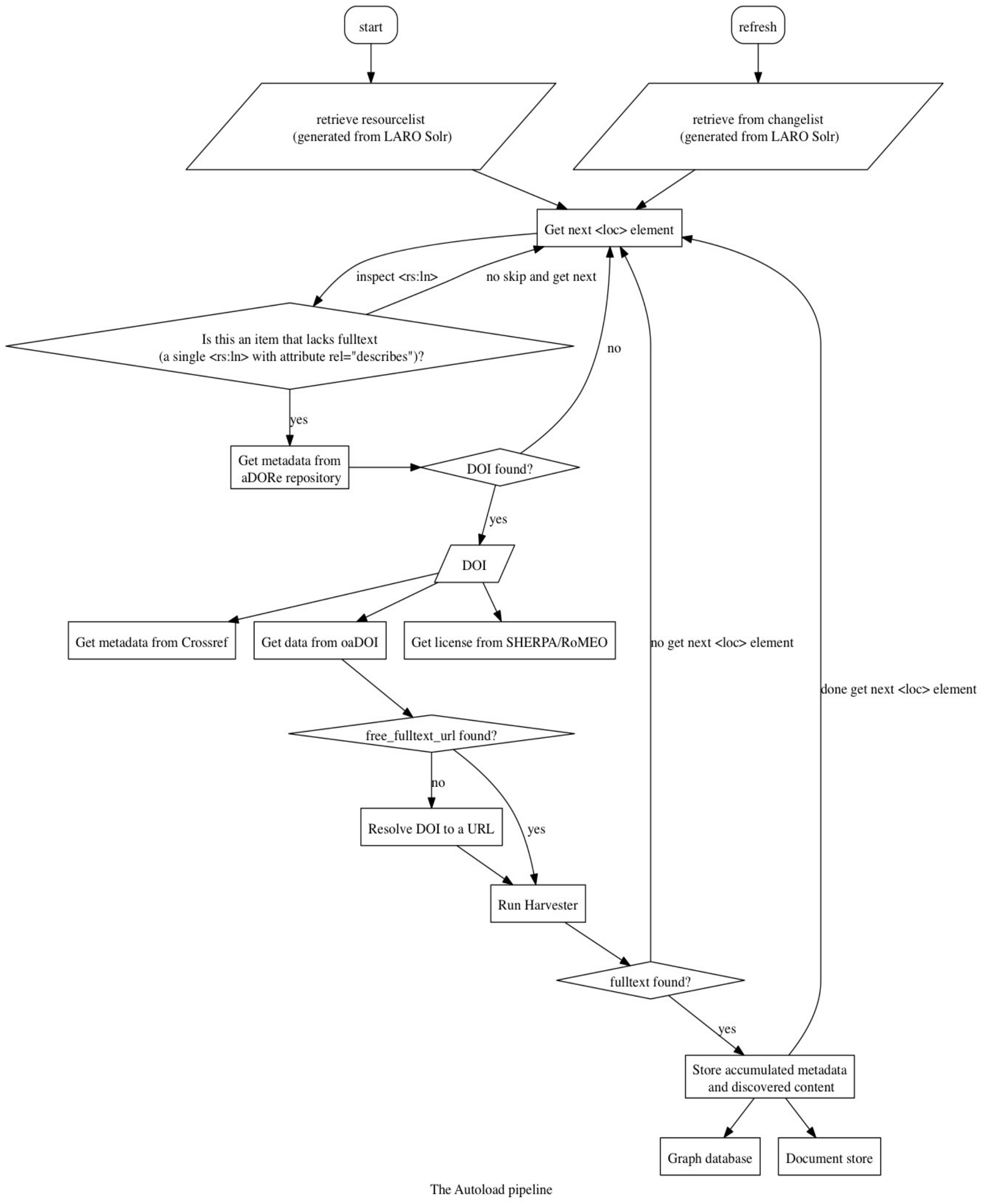

start
refresh
retrieve resourcelist (generated from LARO Solr)
retrieve from changelist (generated from LARO Solr)
Get next <loc> element
inspect <rs:ln>
no skip and get next
Is this an item that lacks fulltext (a single <rs:ln> with attribute rel="describes")?
no
yes
Get metadata from aDORe repository
DOI found?
yes
DOI
Get metadata from Crossref
Get data from oaDOI
Get license from SHERPA/RoMEO
no get next <loc> element
done get next <loc> element
free_fulltext_url found?
no
Resolve DOI to a URL
yes
Run Harvester
fulltext found?
yes
Store accumulated metadata and discovered content
Graph database
Document store
The Autoload pipeline

Figure 1: Autoload pipeline

Using Crossref, oaDOI, SHERPA/Romeo in the Autoload pipeline

An initial execution of the Autoload pipeline discovered 21,798 unique DOIs, most of which lacked full text. With this list, Autoload used several approaches in succession in an effort to discover and retrieve full text of the publication or related content. The pipeline utilizes data from Crossref [15], oaDoi [16], SHERPA/Romeo, and it also attempts direct retrieval of content via DOI resolution and screen scraping. Sometimes it discovers related content, rather than the full text of the publication. These can include supplemental figures, charts, or appendices. This was not expected, but we do intend to archive these items as well. This finding reinforces the particular order in which steps are performed in the Autoload pipeline as compared to other similar efforts. These materials would probably not have been discovered if the SHERPA/RoMEO license categories were applied prior to the discovery phase, rather than afterward. On the other hand it does mean that some content may require human review.

Autoload first uses the DOI to retrieve Crossref metadata for the content. Crossref returns JSON-encoded metadata that describes the content. This metadata includes various bibliographic metadata, as well as links to full text content if known, although in practice this element is rarely populated. Crossref metadata provides semantic information about the object itself. It also provides publisher information. At this point in the process, the Autoload pipeline uses values from Crossref, specifically the ISSN, to query SHERPA/RoMEO to retrieve license information that pertains to the content.

The next step in the Autoload pipeline is to resolve the DOI to a URI. One technique it uses is to request the DOI via the oaDOI Web service (listing 7). oaDOI returns various pieces of information about how to access the object pointed to by a DOI, including, if available, a link to a URI that has been vetted and determined to be a `free_fulltext_uri`. In practice, this URI points to a landing page intended for humans, so it still needs to be parsed. We once again apply the simple recursive anchor parser to this page, which as before, will retrieve and parse a target page if "`pdf`" occurs in the link text, or retrieve and store the target file if "`pdf`" occurs in the link target. We found oaDOI returned `free_fulltext_url` entries for over 5300 items, about 25% of the DOIs in LARO.

```
http://api.oadoi.org/v1/publication/doi/10.1080/21672857.2015.1023120

{
  "results": [
    {
      "_title": " The occurrence of non-pulsating stars in the \u03b3 Dor and
\u03b4 Sct pulsation instability regions: results from Kepler quarter 14\u201317
data ",
      "doi": "10.1080/21672857.2015.1023120",
      "doi_resolver": "crossref",
      "evidence": "oa repository (via base-search.net oa url)",
      "free_fulltext_url": "http://arxiv.org/abs/1502.00175",
      "is_boai_license": false,
      "is_free_to_read": true,
      "is_subscription_journal": true,
      "license": null,
      "oa_color": "green",
      "url": "http://doi.org/10.1080/21672857.2015.1023120"
    }
  ]
}
```
Listing 7: a response from oaDOI identifying access points for a paper

Screen scraping is the Autoload pipeline's technique of last resort but unfortunately for now, it remains a necessary evil. Screen scraping is used for oaDOI responses that contained a free_fulltext_url entry which points at an HTML document rather than a PDF file. It is also used for items for which no full text links were found in Crossref or oaDOI. In the latter instances, the DOI is resolved to a URI, which is then followed using a simple recursive routine. This process uses BeautifulSoup, a Python screen scraping library, to parse HTML content. It iterates through this list, inspecting each link. It examines both the anchor text and the anchor target URI, in an attempt to locate references to PDF. If the anchor text contains a reference to "`pdf`" but the anchor target does not, then it retrieves that page and repeats the process. If the anchor target contains the string "`pdf`", then it retrieve the file it points to and store it locally. As mentioned earlier, this strategy is effective at not only retrieving the primary PDF document if available, but also auxiliary information that an author may have provided, if it is in PDF format.

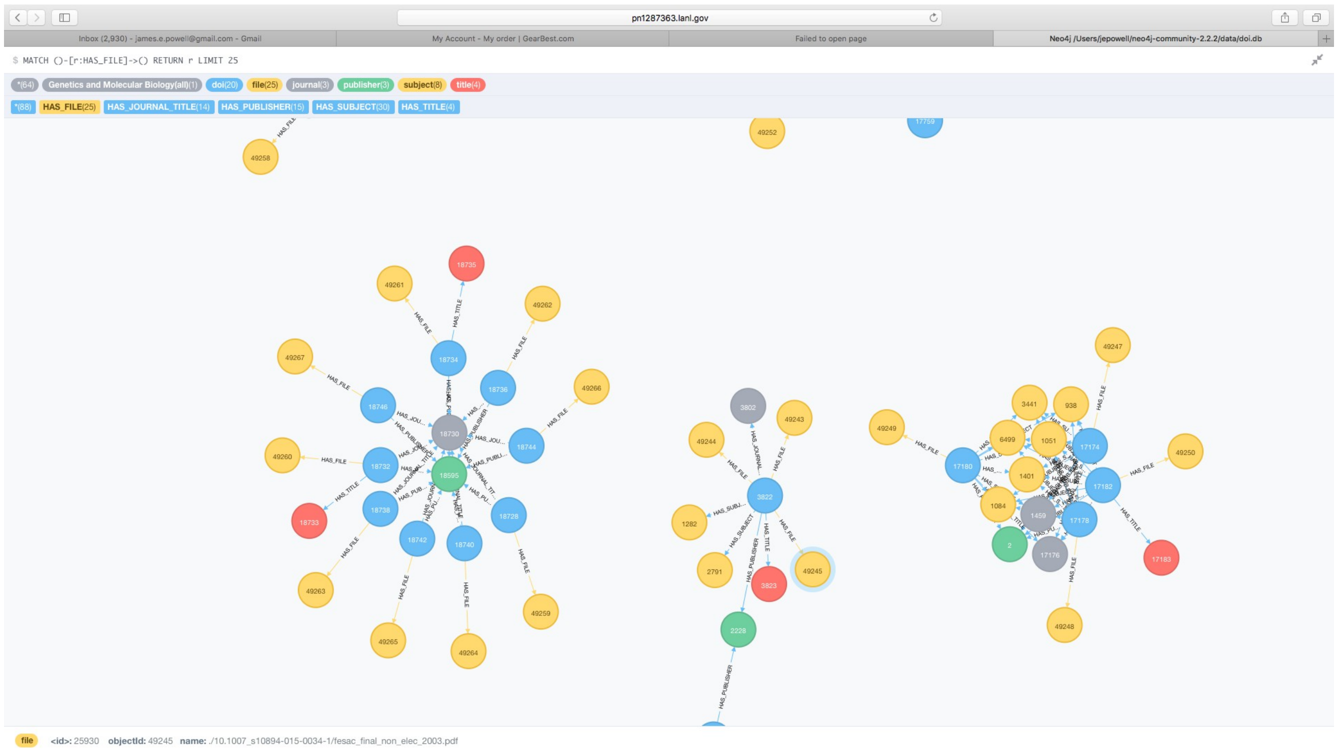
Since we had used Microsoft Academic [17] in a previous project with similar aims, we also evaluated its successor, Microsoft Cognitive Services for use in the Autoload pipeline. The results were tantalizing, yet frustrating. Bing, which appears to be the source of some of the service's data, aggregates links to full text when it is able to disambiguate a title (see example in listing 8). This means that a title search may return more than one full text link for a given paper. However the service had several limitations. DOIs are not indexed so they are not directly searchable. The DOI is considered extended metadata, so it can only be retrieved once a search has been performed on an indexed field such as author or title. Furthermore, partial title searches yielded very poor results for the LARO collection – in fact, a partial title search for a random sample of LARO items yielded zero results. It may be that the content we were searching for had yet to be indexed by Microsoft CS or Bing. Another consideration is that it is a fee-based service. While we plan to keep an eye on this service for future projects, because of these limitations we abandoned plans to incorporate it into the Autoload pipeline.

```
https://api.projectoxford.ai/academic/v1.0/evaluate?expr=Ti='graphs in
libraries'...&model=latest&count=10&offset=0&attributes=Id,E

{
  "expr": "Ti='graphs in libraries'...",
  "entities": [    {
    "E": "{\"DN\":\"Graphs in Libraries: A Primer\",\"D\":\"....\",\"S\":[

     {\"Ty\":3,\"U\":\"http://ejournals.bc.edu/ojs/index.php/ital/article/download/
     1867/1705\"},
   {\"Ty\":1,\"U\":\"https://www.questia.com/library/journal/1G1-272739694/graphs-
     in-libraries-a-primer\"},
     {\"Ty\":1,\"U\":\"http://www.researchgate.net/profile/James_Powell8/publicatio
     n/228776771_Graphs_in_Libraries_A_Primer/links/00b4953500690bfd15000000.pdf\"}
     ,

     {\"Ty\":1,\"U\":\"http://ejournals.bc.edu/ojs/index.php/ital/article/view/1867
     \"}
],\"VFN\":\"Information Technology and
     Libraries\",\"V\":30,\"I\":4,\"FP\":157,\"LP\":169,\"DOI\":\"10.6017/ital.v30i
     4.1867\"}"
    }
  ]
}
```
Listing 8: A request and response example for Microsoft Cognitive Services

When candidate full text is discovered for a DOI, the next step is to analyze this content and assign a match score as compared to its metadata. For this step, Autoload uses the previously retrieved Crossref metadata and compares this to the plain text it extracts from the PDF document. It extracts values from various JSON attributes for the object metadata and subjects this to several data cleaning steps: it tokenizes the content, eliminates stop words, and applies an implementation of the Porter stemming algorithm. The results are a word list containing each unique word that resulted from the data cleaning steps. Next, the pipeline extracts the textual contents of the discovered PDF document. We use a Python library called `PDFMiner` to extract text from the PDF document. This data is also cleaned as per above, but then an additional step is applied: only words that occurred in the metadata word list are included in the document word list. If a word is missing from the latter, then it is included but assigned a value of 0. These are used to evaluate the extent to which the metadata and the discovered document share information. The cosine distance is calculated between the vector representations of the document and the metadata content, results in a similarity score [18]. This score is subsequently associated with the discovered object.

The next step in the Autoload pipeline is retrieval of licensing information from SHERPA/RoMEO.

This service has an easy to use Web services interface and returns an XML document which describes the license associated with a particular ISSN. The SHERPA/RoMEO color is a strong indicator of whether an item can be archived, with green reflecting the licenses which are most amenable to local archiving of some version of the publication. But there are other elements in the response that must be considered. Other license details of interest include information about which version of a publication can be archived, whether there is an embargo period, and additional free text descriptions about the license and the publication. We retrieve this information even for items for which no full text was retrieved, because eventually it may be used to trigger additional discovery steps, for example for discovering preprints.



```
$ MATCH ()-[r:HAS_FILE]->() RETURN r LIMIT 25
```

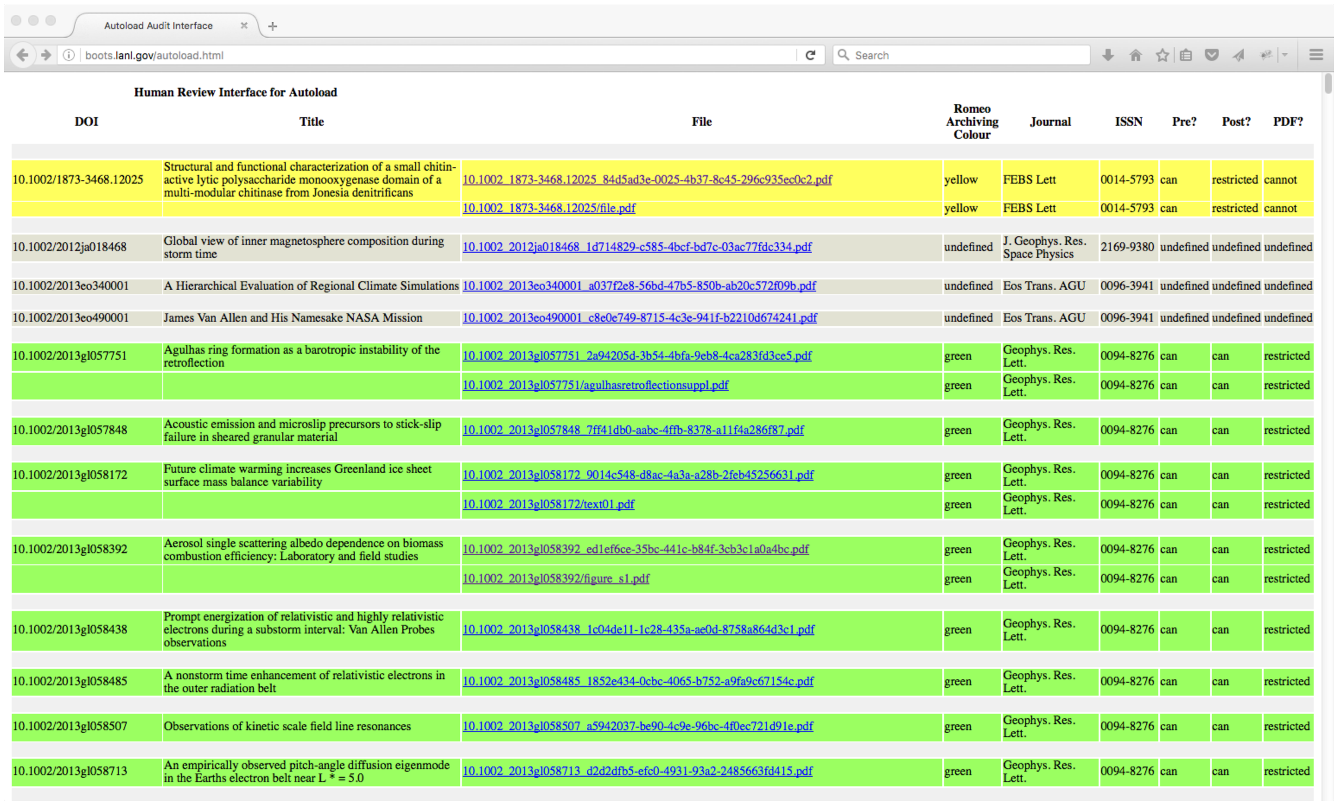
<id>: 25930 objectId: 49245 name: ./10.1007_s10894-015-0034-1/fesac_final_non_elec_2003.pdf

Figure 2: Neo4J graph query visualizer interface

The End of the Line

Autoload aggregates retrieved metadata, licensing information, and details about discovered content in a property graph. This graph has five node types: author, publication, publisher, subject and file. This model is derived from the model utilized in a previous project, described in EgoSystem: Where are our Alumni [19]. The Autoload property graph model differs in that it is concerned with publications, publishers, and discovered content, rather than people, institutions, and publications. Each publisher node is uniquely identified by ISSN, and has associated with it various data from SHERPA/RoMEO. Each file node is uniquely identified by a filename (UUIDs are employed to avoid collisions) and each has a similarity score property with a value computed as described above . Publication nodes are uniquely identified by DOI and have properties that contain values from Crossref. This property graph aggregates all the various information that was discovered for each entry in a resource list or change list that lacked full text. It is intended to facilitate validation of discovered content and verification of its archivability, but other uses have been proposed, such as a review of Crossref as compared with locally assigned subjects.

Populating and updating the graph store is the last step in the pipeline before items can be cleared to be added to LARO. Autoload generates CSV files describing the nodes and relationships for the graph. These files are imported into a Neo4J graph database [20]. It has a visual query and browse interface, as illustrated in figure 2. Neo4J provides a Web services interface which we used to build a simple Web view of harvested content. This interface displays a tabular view of discovered content together with relevant details about each file. Just like the system described in [13] the Web view of harvested content highlights entries using their SHERPA/RoMEO license color code. The Web interface (figure 3) includes the vector similarity score for the object as compared to its metadata, the object's original filename, and a link to the object which a reviewer can access in order to visually inspect its contents. These elements allow a reviewer to make a final determination as to whether this object ought to be added to the repository. When these items are added, they will be indexed by Solr, and subsequently become part of the resource list or change list that the pipeline consumes. These items will be skipped by the Autoload pipeline in future iterations because they were processed successfully in a previous iteration.



Human Review Interface for Autoload

DOI	Title	File	Romeo Archiving Colour	Journal	ISSN	Pre?	Post?	PDF?
10.1002/1873-3468.12025	Structural and functional characterization of a small chitin-active lytic polysaccharide monooxygenase domain of a multi-modular chitinase from Jonesia denitrificans	10.1002_1873-3468.12025_84d5ad3e-0025-4b37-8c45-296c935ec0c2.pdf	yellow	FEBS Lett	0014-5793	can	restricted	cannot
		10.1002_1873-3468.12025/file.pdf	yellow	FEBS Lett	0014-5793	can	restricted	cannot
10.1002/2012ja018468	Global view of inner magnetosphere composition during storm time	10.1002_2012ja018468_1d714829-c585-4bcf-bd7c-03ac77fdc334.pdf	undefined	J. Geophys. Res. Space Physics	2169-9380	undefined	undefined	undefined
10.1002/2013eo340001	A Hierarchical Evaluation of Regional Climate Simulations	10.1002_2013eo340001_a037f2e8-56bd-47b5-850b-ab20c572f09b.pdf	undefined	Eos Trans. AGU	0096-3941	undefined	undefined	undefined
10.1002/2013eo490001	James Van Allen and His Namesake NASA Mission	10.1002_2013eo490001_c8e0e749-8715-4c3e-941f-b2210d674241.pdf	undefined	Eos Trans. AGU	0096-3941	undefined	undefined	undefined
10.1002/2013gl057751	Agulhas ring formation as a barotropic instability of the retroflection	10.1002_2013gl057751_2a94205d-3b54-4bfa-9eb8-4ca283fd3ce5.pdf	green	Geophys. Res. Lett.	0094-8276	can	can	restricted
		10.1002_2013gl057751/agulhasretroflectionsuppl.pdf	green	Geophys. Res. Lett.	0094-8276	can	can	restricted
10.1002/2013gl057848	Acoustic emission and microslip precursors to stick-slip failure in sheared granular material	10.1002_2013gl057848_7ff41db0-aabc-4ffb-8378-a11f4a286f87.pdf	green	Geophys. Res. Lett.	0094-8276	can	can	restricted
10.1002/2013gl058172	Future climate warming increases Greenland ice sheet surface mass balance variability	10.1002_2013gl058172_9014c548-d8ac-4a3a-a28b-2feb45256631.pdf	green	Geophys. Res. Lett.	0094-8276	can	can	restricted
		10.1002_2013gl058172/text01.pdf	green	Geophys. Res. Lett.	0094-8276	can	can	restricted
10.1002/2013gl058392	Aerosol single scattering albedo dependence on biomass combustion efficiency: Laboratory and field studies	10.1002_2013gl058392_ed1ef6ce-35bc-441c-b84f-3cb3c1a0a4bc.pdf	green	Geophys. Res. Lett.	0094-8276	can	can	restricted
		10.1002_2013gl058392/figure_s1.pdf	green	Geophys. Res. Lett.	0094-8276	can	can	restricted
10.1002/2013gl058438	Prompt energization of relativistic and highly relativistic electrons during a substorm interval: Van Allen Probes observations	10.1002_2013gl058438_1c04de11-1c28-435a-ae0d-8758a864d3c1.pdf	green	Geophys. Res. Lett.	0094-8276	can	can	restricted
10.1002/2013gl058485	A nonstorm time enhancement of relativistic electrons in the outer radiation belt	10.1002_2013gl058485_1852e434-0cbc-4065-b752-a9fa9c67154c.pdf	green	Geophys. Res. Lett.	0094-8276	can	can	restricted
10.1002/2013gl058507	Observations of kinetic scale field line resonances	10.1002_2013gl058507_a5942037-be90-4c9e-96bc-4f0ec721d91e.pdf	green	Geophys. Res. Lett.	0094-8276	can	can	restricted
10.1002/2013gl058713	An empirically observed pitch-angle diffusion eigenmode in the Earths electron belt near L * = 5.0	10.1002_2013gl058713_d2d2dfb5-efc0-4931-93a2-2485663fd415.pdf	green	Geophys. Res. Lett.	0094-8276	can	can	restricted

Figure 3: Autoload Web interface

Conclusion

Autoload is implemented in Python, with much of the work handled by three classes: a harvester class, a comparison class and a graph generation class. An initial execution of the pipeline identified and harvested nearly 4,000 objects. As Autoload is a work in progress, we do not yet automatically ingest the objects it discovers, but instead archive them for review. If even half of them were deemed appropriate copies and archivable, Autoload will have tripled LARO's full text holdings. This project motivated the solrSync proof of concept, which can generate ResourceSync resource lists and change lists dynamically. It also serves to demonstrate an extremely useful non-replication use case for ResourceSync: sustaining a workflow that acts upon resource lists and change lists. It points the way forward for developing self-sustaining processes based on ResourceSync. It is likely that we will adapt this approach to support several other efforts, including providing bibliographic metadata for characterizing institutional output and expertise, generating a corpus of locally authored publications for text and data mining (TDM) collection – where the TDM will aggregate and perform additional processing on content from several ResourceSync sources, as well as other more typical data replication use cases, such as exposing LARO to Google for harvesting and indexing. The weakest link in this pipeline is the Autoload harvesting component, because it essentially emulates user interactions and uses screen scraping and brittle heuristics. This situation will hopefully improve as content providers begin to adopt the Signposting model [21]. Signposting uses typed links in HTTP response headers to address some common information discovery patterns, including discovering bibliographic metadata and locating full text from a landing page. This would take much of the guess work out of the process of discovering full text for a resource. The Autoload pipeline would also benefit from the availability of a licensing pattern (RFC5988 includes a "`license`" relation type which could be adapted for this purpose), which might reference a record in SHERPA/RoMEO or perhaps a Creative Commons license, as applicable.

At this point we have come full circle: the Autoload pipeline, initiated by ResourceSync and applied to LARO, generates a list of new content that can be ingested into LARO. As Autoload expands LARO's full text holdings, it is becoming increasingly worth our while to return to our initial goal: using ResourceSync resource lists and change lists to enhance the *discoverability* of content in LARO.

Notes

"A Machine that Would Go of Itself:The Constitution in American Culture" is the title of a book by Michael Kammen.

References

1. Van de Sompel, H, Sanderson, R, Klein, M, Nelson, M.L., Haslhofer, B, Warner, S, Lagoze, C. 2012. A Perspective on Resource Synchronization. D-Lib Magazine [Internet] Volume 18 Number 9/10 [Cited 2017 March 21]. Available from http://www.dlib.org/dlib/september12/vandesompel/09vandesompel.html

2. NISO standard Z39.99.2017 http://www.niso.org/standards/z39-99-2017/

3. Van De Sompel, H., Bekaert, J., Liu, X., Balakireva, L, Schwander, T.. aDORe: A Modular, Standards-Based Digital Object Repository, The Computer Journal, September 2005, Volume 48, Number 5 [Cited 2017 April 20], http://dx.doi.org/10.1093/comjnl/bxh114

4. Apache Solr Reference Guide. 2015. [Cited 2017 Feb 7]. Available from: https://cwiki.apache.org/confluence/display/Solr/Apache+Solr+Reference+Guide

5. Sitemaps XML format. 2016. [Cited 2017 Feb 7]. Available from: https://www.sitemaps.org/protocol.html

6. Klein, M, Van de Sompel, H. 2013. Extending Sitemaps for ResourceSync. JCDL '13. Available from: http://doi.acm.org/10.1145/2467696.2467733

7. Klein, M, Sanderson R, Van de Sompel, H, Warner, S, Lagoze, C, Nelson, M. L. 2013. A Technical Framework for Resource Synchronization. D-Lib Magazine [Internet] Volume 19 Number 1/2 [Cited 2017 Feb 7]. Available from: http://www.dlib.org/dlib/january13/klein/01klein.html

8. Warner, S. resync. 2016. [Cited 2017 Feb 7]. Available from: https://github.com/resync/resync

9. Shankar, H., Klein, M., Van de Sompel, H. ResourceSync PuSH. 2013. [Cited 2017 Apr 20]. Available from https://github.com/resync/resourcesync_push.

10. Shankar, H., Klein, M., Van den Burg, H., rspub-core. 2017. [Cited 2017 Apr 20]. Available from: https://github.com/EHRI/rspub-core

11. Django Documentation. [Cited 2017 Feb 7]; Available from: https://docs.djangoproject.com/en/1.10/

12. Reitz, K. 2016. The Hitchhiker's Guide to Python: XML Parsing. Available from: http://docs.Python-guide.org/en/latest/scenarios/xml/

13. Flynn, S, Oyler, C, Miles, M. 2013. Using XSLT and Google Scripts to Streamline Populating an Institutional Repository. Code4lib [Internet]. [Cited 2017 Feb 7]; Issue 19. Available from: http://journal.code4lib.org/articles/7825

14. SHERPA/RoMEO Applications Programmers' Interface. 2013. [Cited 2017 Feb 7]. Available from: http://www.sherpa.ac.uk/romeo/apimanual.php?la=en

15. Crossref REST API. 2016. [Cited 2017 Feb 7]. Available from: https://github.com/CrossRef/rest-api-doc/blob/master/rest_api.md

16. Introducing oaDOI: Resolving a DOI straight to OA. 2016. Impactstory blog [Internet]. [Cited 2017 Feb 7]. Available from http://blog.impactstory.org/introducting-oadoi/

17. Microsoft Academic Services Documentation. [Cited 2017 Feb 7]; Available from: https://www.microsoft.com/cognitive-services/en-us/documentation

18. Muhlestein, D. 2007. Similarity of Texts: The Vector Space Model with Python. All My Brain blog [Internet]. [Cited 2017 Feb 7]; Available from: http://allmybrain.com/2007/10/19/similarity-of-texts-the-vector-space-model-with-Python/

19. Powell, J, Shankar, H, Rodriguez, R, Van de Sompel, H. 2014. EgoSystem: Where are our Alumni? Code4lib [Internet]. [Cited 2017 Feb 7]; Issue 24. Available from: http://journal.code4lib.org/articles/9519

20. Using the Neo4j REST API. 2012. Hack Sparrow blog [Internet]. [Cited 2017 Feb 7]. Available from: http://www.hacksparrow.com/neo4j-tutorial-rest-api.html

21. Signposting the Scholarly Web [Internet]. [Cited 2017 March 21]. Available from http://signposting.org

About the authors

James Powell is a Research Technologist and a member of the Digital Library Research and Prototyping Team at the Research Library at Los Alamos National Laboratory. His latest book is Powell, James E., and Matthew Hopkins. *A Librarian's Guide to Graphs, Data and the Semantic Web*. Chandos Information Professional Series. Waltham, MA: Chandos, 2015.

Martin Klein is a scientist in the Prototyping Team of the Los Alamos National Laboratory Research Library. His research interests include web-based scholarly communication and system interoperability, temporal aspects of the web, and information retrieval and extraction. He is the lead editor of the ResourceSync specification and is involved in work on Robust Links and Memento.

Herbert Van de Sompel is an information scientist at the Los Alamos National Laboratory and leads the Prototyping Team in the Research Library. The Team does research regarding various aspects of scholarly communication in the digital age. Herbert has played a role in various interoperability efforts (OAI-PMH, OpenURL, OAI-ORE, info URI, Open Annotation, ResourceSync, SharedCanvas, Memento) and in the design of scholarly discovery tools (SFX linking server, bX recommender engine).Currently, he works on the Robust Links effort, aimed at addressing reference rot, and contemplates about Archiving the Web-Based Scholarly Record. More information about Herbert can be found at http://public.lanl.gov/herbertv/